中介者模式

> 中介者模式的作用就是解除对象与对象之间的耦合关系, 增加一个中介者对象后, 所有相关的对象都通过中介者对象来通信, 而不是互相引用, 中介者模式使网状的多对多的关系变成了相对简单的一对多关系

来个多人游戏的实例

普通版

```
function Player( name, teamColor ){
    this.partners = []; // 队友列表
    this.enemies = []; // 敌人列表
    this.state = 'live'; // 玩家状态
    this.name = name; // 角色名字
    this.teamColor = teamColor; // 队伍颜色
};

Player.prototype.win = function(){ // 玩家团队胜利
    console.log( 'winner: ' + this.name );
};

Player.prototype.lose = function(){ // 玩家团队失败
    console.log( 'loser: ' + this.name );
};

Player.prototype.die = function(){ // 玩家死亡
    var all_dead = true;
    this.state = 'dead'; // 设置玩家状态为死亡
    for ( var i = 0, partner; partner = this.partners[ i++ ]; ){ // 遍历队友列表
        if ( partner.state !== 'dead' ){ // 如果还有一个队友没有死亡，则游戏还未失败
            all_dead = false;
            break;
        }
    }
    if ( all_dead === true ){ // 如果队友全部死亡
        this.lose(); // 通知自己游戏失败
        for ( var i = 0, partner; partner = this.partners[ i++ ]; ){ // 通知所有队友玩家游戏失败
            partner.lose();
        }
        for ( var i = 0, enemy; enemy = this.enemies[ i++ ]; ){ // 通知所有敌人游戏胜利
            enemy.win();
        }
    }
};

var playerFactory = function( name, teamColor ){
    var newPlayer = new Player( name, teamColor ); // 创建新玩家
    for ( var i = 0, player; player = players[ i++ ]; ){ // 通知所有的玩家，有新角色加入
        if ( player.teamColor === newPlayer.teamColor ){ // 如果是同一队的玩家
            player.partners.push( newPlayer ); // 相互添加到队友列表
            newPlayer.partners.push( player );
```

```
        }else{
            player.enemies.push( newPlayer ); // 相互添加到敌人列表
            newPlayer.enemies.push( player );
        }
    }
    players.push( newPlayer );
    return newPlayer;
};

//红队：
var player1 = playerFactory( '皮蛋', 'red' ),
    player2 = playerFactory( '小乖', 'red' ),
    player3 = playerFactory( '宝宝', 'red' ),
    player4 = playerFactory( '小强', 'red' );
//蓝队：
var player5 = playerFactory( '黑妞', 'blue' ),
    player6 = playerFactory( '葱头', 'blue' ),
    player7 = playerFactory( '胖墩', 'blue' ),
    player8 = playerFactory( '海盗', 'blue' );

player1.die();
player2.die();
player4.die();
player3.die();
```

现在我们已经可以随意地为游戏增加玩家或者队伍，但问题是，每个玩家和其他玩家都是紧紧耦合在一起的。在此段代码中，每个玩家对象都有两个属性，this.partners和this.enemies，用来保存其他玩家对象的引用。当每个对象的状态发生改变，比如角色移动、吃到道具或者死亡时，都必须要显式地遍历通知其他对象。

在这个例子中只创建了8个玩家，或许还没有对你产生足够多的困扰，而如果在一个大型网络游戏中，画面里有成百上千个玩家，几十支队伍在互相厮杀。如果有一个玩家掉线，必须从所有其他玩家的队友列表和敌人列表中都移除这个玩家。游戏也许还有解除队伍和添加到别的队伍的功能，红色玩家可以突然变成蓝色玩家，这就不再仅仅是循环能够解决的问题了。面对这样的需求，我们上面的代码可以迅速进入投降模式。

中介者模式

```
function Player( name, teamColor ){
    this.name = name; // 角色名字
    this.teamColor = teamColor; // 队伍颜色
    this.state = 'alive'; // 玩家生存状态
};
Player.prototype.win = function(){
    console.log( this.name + ' won ' );
};
Player.prototype.lose = function(){
    console.log( this.name +' lost' );
};
/******************玩家死亡*****************/
Player.prototype.die = function(){
    this.state = 'dead';
    playerDirector.ReceiveMessage( 'playerDead', this ); // 给中介者发送消息，玩家死亡
};
/******************移除玩家*****************/
Player.prototype.remove = function(){
```

```
    playerDirector.ReceiveMessage( 'removePlayer', this ); // 给中介者发送消息，移除一个玩家
};
/********************玩家换队*****************/
Player.prototype.changeTeam = function( color ){
    playerDirector.ReceiveMessage( 'changeTeam', this, color ); // 给中介者发送消息，玩家换队
};

var playerFactory = function( name, teamColor ){
    var newPlayer = new Player( name, teamColor ); // 创造一个新的玩家对象
    playerDirector.ReceiveMessage( 'addPlayer', newPlayer ); // 给中介者发送消息，新增玩家
    return newPlayer;
};

var playerDirector= ( function(){
    var players = {}, // 保存所有玩家
    operations = {}; // 中介者可以执行的操作
    /*****************新增一个玩家***************************/
    operations.addPlayer = function( player ){
        var teamColor = player.teamColor; // 玩家的队伍颜色
        players[ teamColor ] = players[ teamColor ] || []; // 如果该颜色的玩家还没有成立队伍，则
        新成立一个队伍
        players[ teamColor ].push( player ); // 添加玩家进队伍
    };
    /*****************移除一个玩家***************************/
    operations.removePlayer = function( player ){
        var teamColor = player.teamColor, // 玩家的队伍颜色
        teamPlayers = players[ teamColor ] || []; // 该队伍所有成员
        for ( var i = teamPlayers.length - 1; i >= 0; i-- ){ // 遍历删除
            if ( teamPlayers[ i ] === player ){
                teamPlayers.splice( i, 1 );
            }
        }
    };
    /*****************玩家换队***************************/
    operations.changeTeam = function( player, newTeamColor ){ // 玩家换队
        operations.removePlayer( player ); // 从原队伍中删除
        player.teamColor = newTeamColor; // 改变队伍颜色
        operations.addPlayer( player ); // 增加到新队伍中
    };
    operations.playerDead = function( player ){ // 玩家死亡
        var teamColor = player.teamColor,
        teamPlayers = players[ teamColor ]; // 玩家所在队伍
        var all_dead = true;
        for ( var i = 0, player; player = teamPlayers[ i++ ]; ){
            if ( player.state !== 'dead' ){
                all_dead = false;
                break;
            }
        }
        if ( all_dead === true ){ // 全部死亡
            for ( var i = 0, player; player = teamPlayers[ i++ ]; ){
                player.lose(); // 本队所有玩家lose
            }
            for ( var color in players ){
```

```
                if ( color !== teamColor ){
                    var teamPlayers = players[ color ]; // 其他队伍的玩家
                    for ( var i = 0, player; player = teamPlayers[ i++ ]; ){
                        player.win(); // 其他队伍所有玩家win
                    }
                }
            }
        }
    };
    var ReceiveMessage = function(){
        var message = Array.prototype.shift.call( arguments ); // arguments的第一个参数为消息名称
        operations[ message ].apply( this, arguments );
    };
    return {
        ReceiveMessage: ReceiveMessage
    }
})();

// 红队：
var player1 = playerFactory( '皮蛋', 'red' ),
    player2 = playerFactory( '小乖', 'red' ),
    player3 = playerFactory( '宝宝', 'red' ),
    player4 = playerFactory( '小强', 'red' );
// 蓝队：
var player5 = playerFactory( '黑妞', 'blue' ),
    player6 = playerFactory( '葱头', 'blue' ),
    player7 = playerFactory( '胖墩', 'blue' ),
    player8 = playerFactory( '海盗', 'blue' );
player1.die();
player2.die();
player3.die();
player4.die();

// 假设皮蛋和小乖掉线
player1.remove();
player2.remove();
player3.die();
player4.die();

// 假设皮蛋从红队叛变到蓝队
player1.changeTeam( 'blue' );
player2.die();
player3.die();
player4.die();
```